ก่อนจะถึงโอโอพี ท่อนจบ

บุญเลิศ เอี่ยมทัศนา

ในไมโครฯ 53 ผมได้กล่าวถึงการเขียนโปรแกรมในแบบโอโอพีด้วยเทอร์โบปาสคาล 5.5 เรื่องราวของโอโอพีกับเทอร์โบปาสคาลคงมีเพียงเท่าที่กล่าวแล้ว แต่เรื่องราวของเทอร์โบปาสคาล 5.5 ในการสนับสนุนการเขียนโปรแกรมในแบบโอโอพียังไม่จบครับ ที่ขาดอยู่คือการกำหนดนิมิต (ตัวแปร) ให้เป็นแบบไดนามิก เพื่อใช้งานโอโอพีได้กว้างขวางยิ่งขึ้น ดังจะนำมากล่าวในฉบับนี้

นิมิตแบบไดนามิก

การกำหนดนิมิต (ตัวแปรแบบออปเจค) ให้เป็นแบบไดนามิก ดำเนินการเช่นเดียวกับข้อมูลแบบอื่น ๆ คือ กำหนดพอยน์เตอร์เพื่อชี้ไปยังข้อมูลแบบนั้น ๆ อีกทอดหนึ่งและที่ใกล้เคียงกันมากคือข้อมูลแบบเรคอร์ด ดังตัวอย่างในรูปที่ 1

โดยสมมติให้ PersonT เป็นแบบข้อมูลแบบเรคอร์ด ส่วน Circle เป็นแบบข้อมูลแบบออปเจค (ดังกล่าวในไมโครฯ 53) จะได้ว่า FirstPer เป็นตัวแปรพอยน์เตอร์ สำหรับข้อมูลแบบ PersonT และ pcircle เป็นตัวแปรพอยน์เตอร์สำหรับข้อมูลแบบ Circle หรือก็คือนิมิตแบบ Circle นั่นเอง

เมื่อคอมพายเลอร์ดำเนินการมาถึงตรงนี้จะเป็นการกำหนดตัวแปรพอยน์เตอร์ FirstPer และ pcircle ในหน่วยความจำ ส่วนข้อมูลในแบบเรคอร์ดและออปเจคแท้ ๆ จะเกิดขึ้นต่อเมื่อใช้คำสั่ง New และเราสามารถยกเลิกข้อมูลนี้ได้ด้วยคำสั่ง Dispose

ซึ่งในกรณีของออปเจคมีเรื่องที่จะต้องดำเนินการมากกว่า เพื่อความสะดวกแก่ผู้เขียนโปรแกรม เทอร์โบปาสคาล 5.5 จึงได้เตรียมกรรมวิธีและคำไว้ดังจะกล่าวต่อไปนี้

กรรมวิธี

เพื่อเป็นการเปรียบเทียบ จะขอนำกรรมวิธีต่าง ๆ ที่ใช้ในการกำหนดนิมิต

```
TYPE    PerPtrT  = ^PersonT;
        CirPtrT  = ^Circle;
VAR     FirstPer : PerPtrT;
        pcircle  : CirPtrT;
```

รูปที่ 1

มากล่าวรวมกันอีกครั้งคือ

1. เมื่อกำหนดนิมิตในแบบสแตติกและไม่เป็นเวอร์ชวล (คือตามตัวอย่างแรก : ไมโครฯ 53) คอมพายเลอร์จะกำหนดเนื้อที่ในหน่วยความจำส่วนที่เป็นดาต้าเซกเมนต์ สำหรับเก็บบุคลิก (ข้อมูล) และพอยน์เตอร์ (ไปยังโปรแกรมย่อยที่เป็นกิจกรรม) ในลักษณะเดียวกันกับการกำหนดตัวแปรแบบเรคอร์ด

2. เมื่อกำหนดนิมิตในแบบสแตติกและเป็นเวอร์ชวล (คือตามตัวอย่างที่สอง : ไมโครฯ 53) คอมพายเลอร์จะกำหนดเนื้อที่ในหน่วยความจำดังกล่าวในข้อ 1 แต่พอยน์เตอร์จะชี้ไปยังตาราง VMT (Virtual Method Table) ซึ่งจะมีพอยน์เตอร์ไปยังโปรแกรมย่อยที่เป็นกิจกรรมอีกทอดหนึ่ง (เฉพาะกิจกรรมที่กำหนดให้เป็นเวอร์ชวล)

3. เมื่อกำหนดมิติในแบบไดนามิกซึ่งจะต้องกำหนดตัวแปรพอยน์เตอร์ของออปเจคขึ้นก่อน แล้วใช้คำสั่ง New กับตัวแปรนี้เพื่อสร้างนิมิตขึ้นในฮีพ ในการนี้เทอร์โบปาสคาลได้กำหนดไว้ว่าจะต้องเป็นออปเจคแบบเวอร์ชวลเสมอ ทั้งนี้เพราะว่าในตาราง VMT นอกจากจะเก็บพอยน์เตอร์ดังกล่าวมาแล้ว ยังเก็บขนาดของออปเจค (ซึ่งก็คือขนาดของนิมิต) ไว้อีกด้วย

เมื่อดำเนินการตามที่กล่าวมานี้ คอมพายเลอร์จะกำหนดนิมิตดังที่กล่าวในข้อ 2 แต่อยู่ในฮีพแทน

คำที่เพิ่ม

เพื่อให้สามารถดำเนินการได้ดังกล่าวมา ในเทอร์โบปาสคาลได้เพิ่มคำและปรับปรุงคำ ดังนี้คือ

Destructor เป็นคำสงวนที่กำหนดขึ้นใหม่มีฐานะเป็นฟังก์ชันแบบหนึ่ง ซึ่งหากกำหนดให้กิจกรรมใดเป็น Destructor แล้ว เมื่อเรียกใช้กิจกรรมนี้ นอกจากจะทำงานตามโปรแกรมที่กำหนดให้แล้วยังให้ค่าเป็นขนาดของออปเจคนั้นในตาราง VMT กลับมาด้วย (ตามลักษณะของฟังก์ชัน)

หมายเหตุ สำหรับกิจกรรมที่เป็น Constructor และ Destructor หากเราไม่ประสงค์ให้ดำเนินการใดอื่นอีก คงเขียนโปรแกรมย่อยเป็น begin end; เท่านั้น

New ได้ปรับปรุงคำสั่งใหม่ดังนี้

1. ให้เป็นได้ทั้งโพรซีเดอร์และฟังก์ชันโดยที่

- ถ้าต้องการให้เป็นโพรซีเดอร์ ให้ผ่านค่าตัวแปรพอยน์เตอร์ไปให้ เช่น New (pcircle);

- ถ้าต้องการให้เป็นฟังก์ชัน ให้ผ่านค่าเป็นแบบของข้อมูลพอยน์เตอร์ไปให้ เช่น

```
FirstPer : = New(PerPtrT)
pcircle : = New(CirPtrT)
```

2. ให้สามารถผ่านค่ากิจกรรมที่เป็น Constructor (เช่น Init) ไปให้แก่ New เป็นค่าพารามิเตอร์ที่สองได้ด้วย ทั้งเมื่อใช้ New เป็นโพรซีเดอร์หรือฟังก์ชัน เช่น

```
New(pcircle, Init (200, 100, 50));
```

ทั้งนี้เพื่ออำนวยความสะดวกกันไม่ให้ผิดพลาดได้ง่าย ไม่เช่นนั้นจะต้องเขียนโปรแกรมเป็น

```
New (pcircle) ;
pcircle^. Init(200, 100, 50);
```

ซึ่งจะต้องเรียงลำดับตามนี้ด้วย

Dispose ได้ปรับปรุงให้สามารถผ่านค่ากิจกรรมที่เป็น Destructor ไปให้แก่ Dispose เป็นค่าพารามิเตอร์ที่สองได้ด้วย เช่น

```
Dispose(pcircle, Fin);
```

โดยที่ Fin เป็นกิจกรรมที่กำหนดไว้เป็น Destructor ทั้งนี้เพื่ออำนวยความสะดวกเช่นกันไม่เช่นนั้น จะต้องเขียนโปรแกรมเป็น

```
pcircle^. Fin;
Dispose(pcircle);
```

ซึ่งจะต้องเรียงลำดับตามนี้ด้วย โดยที่ Fin จะเป็นผู้ดำเนินการต่าง ๆ ในการยกเลิกมิติ มีรายละเอียดที่จะขอนำไปกล่าวในตัวอย่าง แต่มีเรื่องที่ควรทราบไว้ก่อนคือ

- การกำหนดกิจกรรมเป็น Destructor ไม่เกิดผลเสียใดถึงแม้จะไม่ใช้ (คือจะไม่กำหนดมิติในแบบไดนามิก) แต่เป็นผลดีคือพร้อมให้ใช้เสมอ

- กิจกรรมที่เป็น Destructor จะต้องใช้อยู่ใน Dispose เท่านั้น

- ควรให้เป็น Virtual เพื่อให้สามารถยกเลิกข้อมูลแบบไดนามิกซ้อนไดนามิกได้ (ดังจะแสดงในตัวอย่าง)

ก่อนจะไปต่อ

จากฉบับที่แล้วผมได้สร้างยูนิต PICT0 ไว้ (สำหรับของผมจะเป็น PICT02) ขอให้ดูในรูป D1 ซึ่งเป็นการกำหนดออปเจค Gwindow ผมได้แอบใส่กิจกรรม Fin ไว้เป็นแบบ destructor เพื่อจะนำมากล่าวถึงในฉบับนี้

แต่ผมลืมไปเรื่องหนึ่งคือ ผมได้ให้ Fin เป็นกิจกรรมในแบบเวอร์ชวล นั่นคือ บรรพบุรุษของมันคือ Point จะต้องมีกิจกรรม Fin ที่เป็น destructor และเป็นเวอร์ชวลด้วย (ผมแก้ของผมแต่ลืมแจ้งให้คุณผู้อ่านทราบ ขออภัยด้วยครับ) ขอให้แก้ออปเจค Point เป็นดังรูปที่ 2 และป้อนโปรแกรมย่อยในส่วน Implementation ดังรูปที่ 3 ด้วยครับ

อีกเรื่องหนึ่งคือผมได้กำหนดค่าคงตัวดังแสดงในรูปที่ 4 ไว้ในยูนิต GRP1 แต่ไม่ได้แจ้งให้ทราบก็ต้องขออภัยอีกเช่นกันครับ

นิมิตไดนามิกธรรมดา

โปรแกรม ExampleFive (รูปที่ 5) แสดงถึงการกำหนดนิมิตในแบบได-

```
Point = OBJECT(Locate)
            quit : Boolean;
constructor Init(fx,fy : Integer);
destructor  fin;   VIRTUAL;
procedure   Hide;  VIRTUAL;
procedure   Show;  VIRTUAL;
procedure   JumpTo(nx,ny : Integer);
procedure   Drag(dist : Integer);
function    IsQuit : Boolean;
          END;
```

รูปที่ 2

```
DESTRUCTOR  Point.Fin;
begin  Hide   end;
```

รูปที่ 3

```
CONST   CR = #13;  LF = #10;  BS = #8;  Esc = #27;  SPACE = ' ';
        up = #72;  down = #80;  left = #75;  right = #77;
        home = #71;  endd = #79;  PgUp = 73;  PgDn = #81;
VAR     ch : Char;
```

รูปที่ 4

นามิก อย่างธรรมดา คือเช่นเดียวกับข้อมูลแบบอื่น ๆ ได้ผลการรันเมื่อจบโปรแกรมดังรูปที่ 6 ในโปรแกรมนี้ได้ให้ดำเนินการตามลำดับดังนี้คือ

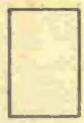
- ให้แสดงขนาดของฮีพ
- กำหนดนิมิตในแบบไดนามิก ซึ่งได้แสดงวิธีดำเนินการทั้งสองแบบ
- ให้แสดงขนาดของฮีพที่เหลือ
- ให้ตัวละครแสดงตัว จะเห็นเป็นรูปวงกลมและสี่เหลี่ยม
- ให้ยกเลิกนิมิตในฮีพ ซึ่งสำหรับสี่เหลี่ยมให้ดำเนินการเช่นเดียวกับตัวแปรแบบอื่น ๆ ส่วนวงกลมได้ใช้ destructor ใน Dispose ทำให้ภาพของวงกลมหายไปเพราะคำสั่ง Hide ใน Point.Fin (รูปที่ 3)
- เสร็จแล้วให้แสดงขนาดของฮีพอีกครั้งจะเห็นได้ว่าจำนวนไบต์เท่ากับเมื่อเริ่มต้น นั่นคือในกรณีของนิมิตไดนามิกธรรมดา เราจะใช้ destructor ใน Dispose หรือไม่ก็ได้ แต่ควรใช้เพราะได้ออกแบบมาให้ใช้

อีกเรื่องหนึ่งคือตามคู่มือของเทอร์โบปาสคาล 5.5 ได้บ่งไว้ว่ากิจกรรมใดที่กำหนดให้เป็นเวอร์ชวลจะต้องเป็นเวอร์ชวลตลอด ในกรณีนี้คือ Fin ได้กำหนดให้เป็นเวอร์ชวลไว้ใน Point ไม่ได้กำหนดใน Circle แต่กลับสามารถใช้ของ Point ได้คือที่อยู่ใน

```
{ EXP5.PAS }

PROGRAM  ExampleFive;
uses   crt,graph,grpl,pict02;
TYPE   CptrT = ^Circle;     RptrT = ^Rectangle;
VAR    cirptr : CptrT;      recptr : RptrT;


BEGIN
  OpenGraph;

  DispHeapStatus('Heap space before dynamic allocation ');
  New(cirptr);  cirptr^.Init(100,100,50);
  recptr := New(RptrT,Init(200,100,50,50));
  DispHeapStatus('Heap space after allocation ');

  cirptr^.Show;   recptr^.Show;

  ReadLn;
  Dispose(recptr);  Dispose(cirptr,fin);
  DispHeapStatus('After dispose ');

readln;  CloseGraph
END.
```

รูปที่ 5

```
Heap space before dynamic allocation 151763
Heap space after allocation 151743
After dispose 151763
```



รูปที่ 6
ผลการรันของโปรแกรม ExampleFive

คำสั่ง Dispose

นิมิตไดนามิกซ้อน

แต่ในกรณีที่กำหนดข้อมูลแบบไดนามิกไว้ในออปเจค แล้วนำออปเจคมากำหนดเป็นแบบไดนามิกอีก จะต้องใช้ destructor ใน Dispose เสมอ ดังโปรแกรม ExampleSix (รูปที่ 7) กล่าวคือ

ในการกำหนดออปเจค Gwindow

(รูป D1 และ D2 ในไมโครฯ 53) จุดประสงค์คือต้องการใช้เป็นหน้าต่างที่เคลื่อนไหวไปมาบนจอภาพได้ จึงจำเป็นต้องเก็บภาพเดิมตรงที่จะนำหน้าต่างไว้ใน old และภาพในกรอบหน้าต่างของตนไว้ใน newone แต่ขณะที่เราเขียนโปรแกรมของออปเจค Gwindow เราไม่ทราบขนาดของหน้าต่าง จึงกำหนดตัวแปร 2 ตัวนี้ไว้เป็นพอยน์เตอร์

และให้ Gwindow. Init จองเนื้อที่ในฮีพด้วยคำสั่ง GetMem ตามขนาดของหน้าต่างที่ผู้เขียนโปรแกรมในระดับผู้กำกับจะผ่านค่ามาให้แก่ x1, y1, x2, และ y2

หากต่อไปผู้กำกับไม่ต้องการใช้หน้าต่างนี้และอาจต้องการเนื้อที่ในฮีพไปใช้งานอื่น จะยกเลิกได้ด้วย Gwindow.Fin (ซึ่งจะลบหน้าต่างนี้ด้วย) ซึ่งผู้กำกับจะทำได้ด้วยใช้ Fin ใน Dispose คู่กับตัวละครดังที่เห็นในโปรแกรม ExampleSix

เมื่อดำเนินการเช่นนี้ผู้กำกับจะไม่ต้องพะวงเรื่องอื่นใดเลย เพียงแต่ใช้ Init ใน New และใช้ Fin ใน Dispose ควบคู่กับตัวละครที่ต้องการ จะยกเลิกหน่วยความจำในฮีพได้อย่างหมดจด

ขอให้ทดลองตัด Fin ใน Dispose ออกจะพบว่าได้ขนาดของฮีพน้อยกว่าเมื่อเริ่มแรก ซึ่งหน่วยความจำที่เสียไปนี้จะเป็นการเสียเปล่าไม่สามารถนำมาใช้ได้อีก (ในการรันครั้งนั้น)

```
{ EXP6.PAS  }

PROGRAM  ExampleSix;
uses    crt,graph,grp1,pict02;
TYPE    GwndPtrT  = ^Gwindow;
VAR     info,memo : GwndPtrT;

PROCEDURE   MainWindow;
begin  SetViewPort(0,0,GetMaxX,GetMaxY,clipon)  end;

BEGIN
  OpenGraph;

  DispHeapStatus('Before allocate ');
  New(info,Init(0,120,400,150));
  New(memo,Init(500,100,600,150));
  DispHeapStatus('After allocate ');


  WITH  info^  DO
    begin  WindowOn;
           OutText('  Turbo Pascal 5.5 gives you the power of ');
           OutText('Object-Oriented Programming');
    end;


  ReadLn;
  Dispose(memo,fin);  Dispose(info,fin);
  MoveTo(0,24);
  DispHeapStatus('After Dispose ');

Graph.OutTextXY(10,36,'End of program press ENTER');

readln;  CloseGraph
END.
```

รูปที่ 7

A

```
{ EXP7.PAS }

PROGRAM  ExampleSeven;
uses    crt,graph,grp1,pict02;
TYPE    PptrT = ^Point;  CptrT = ^Circle;
        RptrT = ^Rectangle;

        NodePtrT = ^NodeT;
        NodeT    = RECORD
          item : PptrT;
          next : NodePtrT;
                 END;

        ListPtrT = ^ListT;
        ListT    = OBJECT
          node : NodePtrT;
          constructor Init;
          destructor  Done; VIRTUAL;
          procedure   Add(item : PptrT);
          procedure   Report;
                END;
```

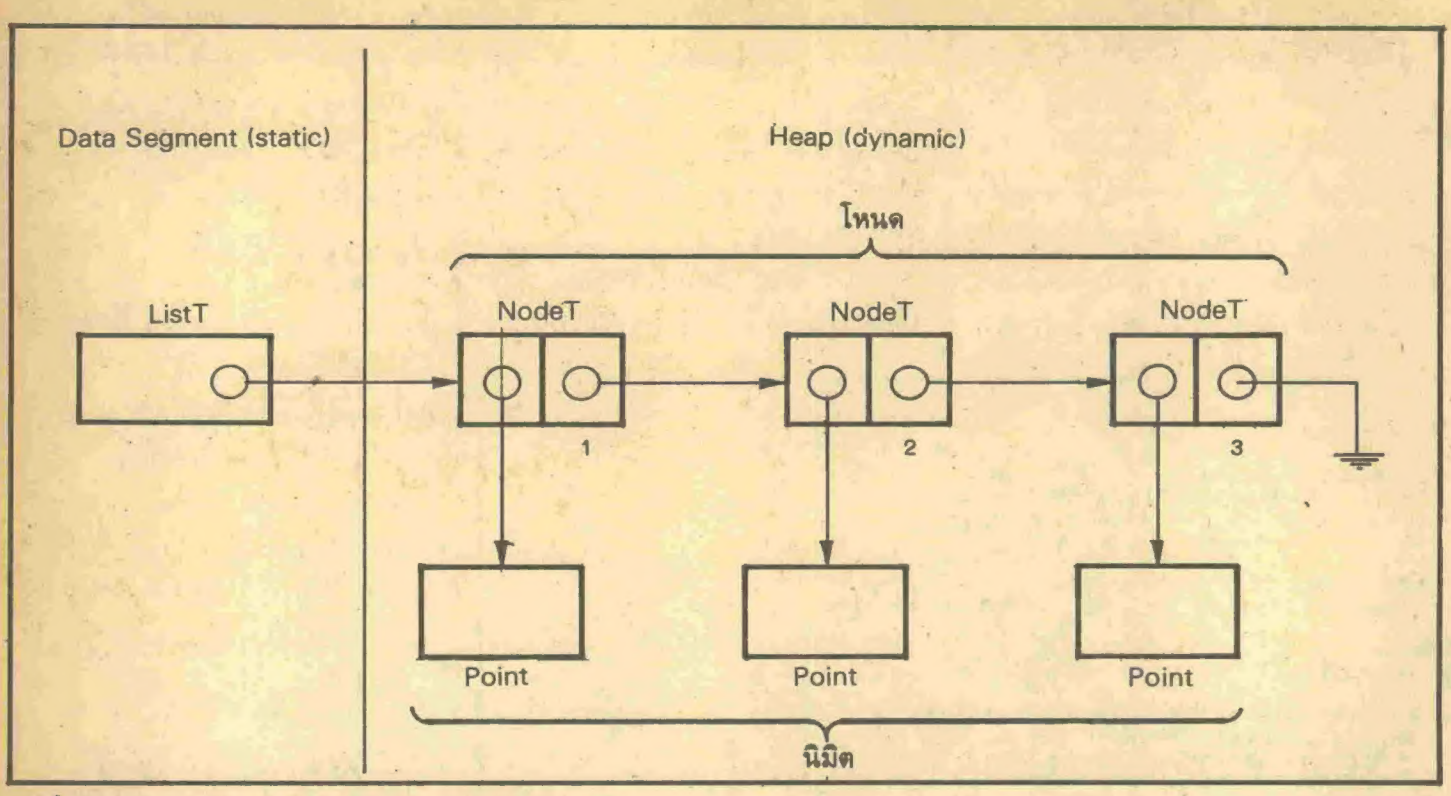
นิมิตแบบลิงค์ลิสต์

ในบางกรณีผู้เขียนโปรแกรมระดับผู้กำกับอาจต้องการกำหนดนิมิตในแบบลิงค์ลิสต์ (เช่นเดียวกับที่ดำเนินการกับข้อมูลแบบเรคอร์ด) ปัญหาคือเราไม่ได้กำหนดฟิลด์พอยน์เตอร์ไว้ในออปเจคดังกล่าวมา คือใน Point และสมมติว่าไม่มีซอร์สโค้ดของยูนิต PICT0

แต่การเขียนโปรแกรมในแบบโอโอพีต้องการให้ต่อเติมเสริมแต่งออปเจคได้ตามต้องการ ซึ่งเราจะทำได้โดยกำหนดแบบข้อมูลพอยน์เตอร์ของเรคอร์ดครอบออปเจค ซึ่งในกรณีนี้คือ Point อีกชั้นหนึ่ง

ดังแสดงในรูป A ซึ่งเป็นส่วนหนึ่งของโปรแกรม ExampleSeven ได้กำหนดแบบข้อมูล NodePtrT เป็นแบบข้อมูลพอยน์เตอร์ดังกล่าว ซึ่งจะนำไปสร้างลิงค์ลิสต์ของนิมิตได้โดยมีฟิลด์ item เป็นพอยน์เตอร์ไปยังออปเจคแบบ Point และฟิลด์ next เป็นพอยน์เตอร์ไปยังเรคอร์ดแบบ NodeT ตัวต่อไป ดังแสดงในซีกขวาของรูปที่ 8

การดำเนินการกับลิงค์ลิสต์ปกติจะต้องเตรียมโปรแกรมสำหรับการแทรก



รูปที่ 8 แสดงการจัดเก็บนิมิตในหน่วยความจำ

B

```
CONSTRUCTOR  ListT.Init;
begin  node := nil  end;


DESTRUCTOR  ListT.Done;
VAR      tem : NodePtrT;
BEGIN
  WHILE  node <> nil  DO
    begin  tem := node;
           Dispose(tem^.item, Fin);
           node := tem^.next;
           Dispose(tem)     end;
END;


PROCEDURE  ListT.Add(any : PptrT);
VAR     tem : NodePtrT;
BEGIN
  New(tem);    tem^.item := any;
  tem^.next := node;    node := tem;
END;


PROCEDURE  ListT.Report;
VAR    tem : NodePtrT;  astr : String;
BEGIN
  tem := node;
  WHILE  tem <> nil  DO
    begin  tem^.item^.Show;
           Str(tem^.item^.GetX:3, astr);
           OutText('x = ' + astr);
           Str(tem^.item^.GetY, astr);
           OutText('  y = ' + astr);
           MoveTo(0,GetY+12);
           tem := Tem^.next;       end;
END;
```

และถอนข้อมูลจากลิสต์และเนื่องจากเป็นลิงค์ลิสต์ของออปเจคจึงจะต้องมีโปรแกรมสำหรับถอนลิงค์ลิสต์ทั้งหมดด้วย โปรแกรมเหล่านี้ควรรวมไว้ด้วยกันที่เหมาะสมที่สุดคือทำเป็นออปเจคดังที่กำหนดเป็นออปเจค ListT ในรูป A

ส่วนที่สัมพันธ์กับ NodeT คือฟิลด์ node ใน ListT ดังแสดงในรูปที่ 8 จะเห็นว่าส่วนของ ListT จะอยู่ในหน่วยความจำที่เป็นสแตติก ส่วน NodeT อยู่ในหน่วยความจำที่เป็นไดนามิก (แบบข้อมูล ListPtrT ไม่ได้ใช้ในโปรแกรมนี้ จะกล่าวในตอนท้าย)

กิจกรรมที่กำหนดไว้ใน ListT รูป B จะมีการดำเนินการดังนี้

Init

เพียงแต่ให้ node เป็น nil

Add

เพื่อแทรก NodeT ในลิงค์ลิสต์ การแทรกจะเป็นการแทรกหน้า คือข้อมูลตัวที่แทรกหลังสุดจะเป็นตัวแรกในลิงค์ลิสต์ ขอให้สังเกตว่าได้ให้ item ชี้ไปที่ any ตามแต่โปรแกรมหลักจะ

ผ่านมา ซึ่งอาจเป็น Point หรือ Circle หรือ Rectangle หรือ Gwindow อันใดอันหนึ่งก็ได้ ซึ่งโปรแกรมหลักจะต้องเตรียมการมาให้เรียบร้อยแล้ว

ในโปรแกรมนี้ไม่ได้แสดงกิจกรรมที่จะถอนข้อมูลออกจากลิงค์ลิสต์ (คือที่ดำเนินการตรงข้ามกับ Add)

Report

จุดประสงค์ของกิจกรรมนี้ก็เพื่อแสดงการดำเนินการนิมิตในลิงค์ลิสต์เท่านั้น โดยให้แสดงตัวและแสดงค่า x และ y ของแต่ละนิมิต จากต้นลิสต์ไปท้ายลิสต์เท่านั้นอีกเช่นกันครับ

Done

เพื่อยกเลิกหน่วยความจำของลิงค์ลิสต์นี้ในฮีพทั้งหมด โดยยกเลิกทีละโหนด ขอให้สังเกตการใช้ Fin ใน Dispose ซึ่งถ้าหากไม่มีฟังก์ชันแบบ destructor คือ Fin ที่เป็นผู้ให้ขนาดของนิมิตแก่ Dispose แล้ว จะยกเลิกหน่วยความจำในฮีพที่เป็นนิมิต เช่น Point ไม่ได้ สำหรับคำสั่ง Dispose (tem) เป็นการยกเลิกโหนดในฮีพ เช่นข้อมูลแบบเรคอร์ดโดยทั่วไป

การทำงานโดยส่วนรวมคือจะยกเลิกโหนดทีละโหนด โดยเริ่มจากโหนดหมายเลข 1 (ในรูป) ด้วยการยกเลิกนิมิตก่อนแล้วยกเลิกตัวโหนด เสร็จแล้วไปดำเนินการเช่นเดียวกันนี้กับโหนดหมายเลข 2 และต่อ ๆ ไปจนสุดลิงค์ลิสต์

หากนิมิตใดมีข้อมูลในฮีพอีก (เช่น Gwindow) กิจกรรม Fin ของนิมิตนั้นจะเป็นผู้ยกเลิกก่อนยกเลิกตัวนิมิต (ซึ่งเราจะต้องเขียนโปรแกรมไว้ดังเช่นใน Gwindow.Fin) ด้วยวิธีการเช่นนี้ทำให้เราสามารถยกเลิกหน่วยความจำในฮีพได้อย่างหมดจด

โปรแกรมหลัก

ในโปรแกรมหลักรูป C ขอให้สังเกตการกำหนดตัวแปรด้วย (สำหรับ listptr ไม่ได้ใช้ในโปรแกรมนี้) เริ่มโดยให้แสดงขนาดของฮีพ แล้วเตรียมค่าใน actlist คือให้พอยน์เตอร์เป็น nil

จากนี้เป็นเรื่องของผู้กำกับที่จะกำหนดตัวละครกี่ตัว ในโปรแกรมนี้ได้กำหนดไว้ 3 ตัวด้วยคำสั่ง Add ขอให้สังเกตการใช้ New ควบคู่กับ Init จะเป็นการจองหน่วยความจำในฮีพตามขนาดของออปเจค เกิดเป็นนิมิตขึ้นแล้วและให้ Init เตรียมค่าในนิมิตนั้นให้เสร็จแล้วจะส่งค่าแอดเดรสของนิมิตนี้ไปให้แก่ any ใน ListT.Add ดำเนินการตามที่กล่าวมา

หลังจากกำหนดตัวละคร 3 ตัวแล้วได้ให้ Report ทำงานตามที่กล่าวมาแล้วให้แสดงขนาดของหน่วยความจำในฮีพที่เหลือ ดังผลการรันในรูปที่ 9 แล้วให้ยกเลิกตัวละครในฮีพทั้งหมดด้วย actlist.Done ดังกล่าวแล้ว และให้แสดงขนาดของหน่วยความจำในฮีพอีกครั้ง ซึ่งจะต้องเท่ากับครั้งแรก

เมื่อรันโปรแกรมถึงขั้นตอนนี้จะคงเหลือตัวแปร actlist ในหน่วยความจำสแตติก (อยู่ 8 ไบต์) หากต้องการให้เป็นตัวแปรแบบไดนามิกอีก สามารถดำเนินการได้โดยใช้ตัวแปรพอยน์เตอร์ listPtr ซึ่งต้อง New แต่เริ่มแรก และเมื่อต้องการยกเลิกหน่วยความจำในฮีพทั้งหมดให้ใช้คำสั่งเป็น

Dispose (listPtr, Done);

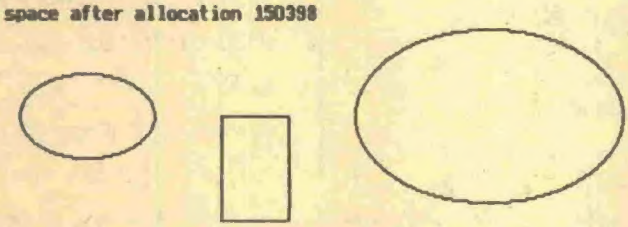
แต่ถึงอย่างไรก็ดีก็ยังเหลือตัวแปรพอยน์เตอร์ listPtr ในหน่วยความจำสแตติกอยู่ 4 ไบต์

ผมคิดว่าเรื่องราวของเทอร์โบปาสคาล 5.5 ในส่วนที่เกี่ยวกับโอโอพีคงมีเท่านี้ครับ

C

```
VAR    actlist : ListT;    listptr : ListPtrT;

BEGIN   { MAIN }
  OpenGraph;

  DispHeapStatus('Heap space before dynamic allocation ');
  WITH  actlist  DO
    begin  Init;
           Add(New(CptrT, Init(100,100,50)));
           Add(New(RptrT, Init(200,100,50,50)));
           Add(New(CptrT, Init(400,100,100)));
           Report   end;

  DispHeapStatus('Heap space after allocation ');
  ReadLn;
  actlist.Done;
  DispHeapStatus('After dispose ');

readln;  CloseGraph
END.
```

```
Heap space before dynamic allocation 150451
x = 400  y = 100
x = 200  y = 100
x = 100  y = 100
Heap space after allocation 150398
```



รูปที่ 9 ผลการรันของโปรแกรม ExampleSeven ก่อนยกเลิกตัวละคร